Tetrahedral and hexahedral mesh adaptation for CFD problems*

Rupak Biswas[a,†], **Roger C. Strawn**[b]

[a] *MRJ Technology Solutions, MS T27A-1, NASA Ames Research Center, Moffett Field, CA*

[b] *US Army AFDD, MS 258-1, NASA Ames Research Center, Moffett Field, CA*

Abstract

This paper presents two unstructured mesh adaptation schemes for problems in computational fluid dynamics. The procedures allow localized grid refinement and coarsening to efficiently capture aerodynamic flow features of interest. The first procedure is for purely tetrahedral grids; unfortunately, repeated anisotropic adaptation may significantly deteriorate the quality of the mesh. Hexahedral elements, on the other hand, can be subdivided anisotropically without mesh quality problems. Furthermore, hexahedral meshes yield more accurate solutions than their tetrahedral counterparts for the same number of edges. Both the tetrahedral and hexahedral mesh adaptation procedures use edge-based data structures that facilitate efficient subdivision by allowing individual edges to be marked for refinement or coarsening. However, for hexahedral adaptation, pyramids, prisms, and tetrahedra are used as buffer elements between refined and unrefined regions to eliminate hanging vertices. Computational results indicate that the hexahedral adaptation procedure is a viable alternative to adaptive tetrahedral schemes.

*To appear in Applied Numerical Mathematics Journal.

†Corresponding author. E-mail: rbiswas@nas.nasa.gov. Work was supported by NASA under Contract Number NAS 2-14303.

1 Introduction

Anisotropic mesh adaptation is a powerful tool for computing steady and unsteady three-dimensional problems that require local grid modifications to efficiently resolve solution features. A number of such methods have recently been successfully developed for local refinement and coarsening of unstructured tetrahedral meshes [6, 9, 10, 11, 16]. However, repeated anisotropic subdivision can significantly deteriorate the quality of a tetrahedral mesh. Previous work [7] has demonstrated that isotropic subdivision is required if mesh quality is to be controlled effectively for arbitrary refinement levels in tetrahedral meshes, without resorting to local mesh regeneration. This is a serious limitation when directional flowfield features are present, leading to an inefficient distribution of grid points in the final mesh. In addition, truly anisotropic subdivision is almost impossible to realize for real problems on tetrahedral meshes.

A remedy for this drawback is to use hexahedral elements which are much better suited for anisotropic subdivision without mesh quality problems [5, 12, 17, 20]. Furthermore, hexahedral meshes also yield more accurate flowfield solutions than their tetrahedral counterparts for the same number of edges [1]. However, hexahedral adaptation schemes generate "hanging" vertices when a hexahedron cannot be split into smaller hexahedra without propagating the mesh refinement into regions where it is not desired.

A brief description of our edge-based tetrahedral adaptation procedure is given in Section 2. Some of its limitations are highlighted in Section 3. Section 4 presents our extension to hexahedral meshes. The hexahedral adaptation scheme shares much of the logic and data structure with the tetrahedral procedure. It, therefore, also shares much of the software. A comparison of the two adaptation schemes in terms of the computer resources required is given in Section 5. Finally, some computational results for three-dimensional airfoils and a non-lifting rectangular helicopter rotor blade in hover are presented in Section 6.

2 Tetrahedral adaptation scheme

Biswas and Strawn [6] describe the tetrahedral mesh adaptation scheme that is used as a framework for the new hexahedral mesh adaptation procedure. The code, called 3D_TAG, has its data structure based on edges that connect the vertices of a tetrahedral mesh. This means that the elements are represented by their six edges rather than by their four vertices. This edge data structure facilitates efficient refinement and coarsening and contains the right amount of information to rapidly reconstruct the mesh connectivity when vertices are added or deleted while having a reasonable memory requirement.

The edge data structure also makes the mesh adaptation compatible with the unstructured-grid CFD flow solver [18] that is used for the numerical calculations in this paper. This Euler solver is a rotary-wing version of the three-dimensional finite-volume upwind method originally developed by Barth [4]. It solves for the solution variables at the vertices of the mesh and satisfies the integral conservation laws on non-overlapping polyhedral control volumes surrounding these vertices. Improved accuracy is achieved by using a piecewise linear reconstruction of the solution in each control volume. An explicit algorithm with local time stepping is used to speed convergence to steady-state results.

At each mesh adaptation step, individual edges are marked for coarsening, refinement, or no change, based on an error indicator calculated from the flow solution. Edges whose error values exceed a user-specified upper threshold are targeted for subdivision. Similarly, edges whose error values lie below another user-specified lower threshold are targeted for removal. Only three subdivision types are allowed for each tetrahedral element and these are shown in Fig. 1. The 1:8 isotropic subdivision is implemented by adding a new vertex at the mid-point of each of the six edges. The 1:4 and 1:2 subdivisions are used in two ways. First, they can result because the edges of a parent tetrahedron are targeted anisotropically. Second, they are used as buffers between the refined elements and the surrounding coarser grid. These buffer elements are required to form a valid connectivity for the new mesh so that there are no "hanging" vertices.

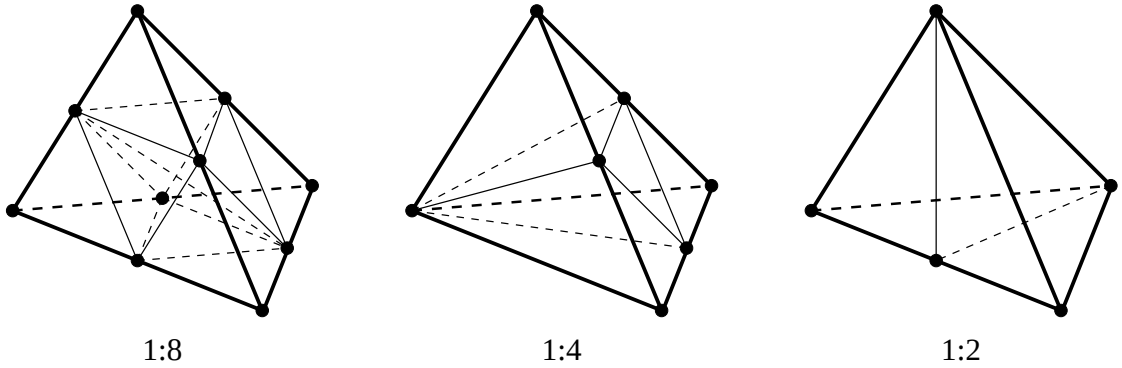


Figure 1: Types of subdivision that are permitted for a tetrahedral element.

Mesh refinement is performed by first setting a bit flag to one for each edge that is targeted for subdivision. The edge markings for each element are then combined to form a binary pattern. Elements are continuously upgraded to valid patterns corresponding to the three allowed subdivision types until none of the edge patterns show any change. Once this edge marking is completed, each element is independently subdivided based on its binary pattern. Special data structures are used in order to ensure that this process is computationally efficient.

Mesh coarsening is also performed using the edge-marking patterns. If a child element has any edge marked for coarsening, this element and its siblings are removed and their parent element is reinstated. The parent edges and elements are retained at each refinement step so they do not have to be reconstructed. Reinstated parent elements have their edge-marking patterns adjusted to reflect that some edges have been coarsened. The parents are then subdivided based on their new patterns by invoking the mesh refinement procedure. As a result, the coarsening and refinement procedures share much of the same logic.

There are several constraints for mesh coarsening. First, edges cannot be coarsened beyond the initial mesh. If that were permitted, some mesh regeneration procedure would have to be used. Second, edges must be coarsened in an order that is reversed from the one by which they were refined. This is a reasonable restriction that allows only certain edges to coarsen at each mesh adaptation step. Third, an edge can coarsen if and only if its sibling is also targeted for coarsening. This guarantees that the parent edge can be reinstated by

removing its center vertex without unacceptably compromising the solution quality. Finally, edges of non-leaf elements or of their siblings (even if the siblings are leaf elements) cannot be coarsened.[1] If this were allowed, it would indirectly violate the requirement that edges coarsen in an order reversed from the one by which they were created.

A significant feature of this adaptation scheme is using reverse pointers for the various mesh entities. For example, not only is every edge in the mesh described as a pair of vertices, but every vertex has a list of all the edges that are incident upon it. Similarly, in addition to defining each tetrahedral element in terms of its six edges, each edge has a list of elements that share it. These reverse pointers eliminate extensive searches and are crucial to the efficiency of the overall mesh adaptation procedure.

3 Limitations of tetrahedral schemes

One of the problems with anisotropic subdivision of tetrahedral meshes is that repeated refinement can lead to poor mesh quality. Poor mesh quality is defined as a grid deficiency that leads to an inaccurate flowfield solution. Poor meshes can have disparate element sizes, large face angles, and high vertex degrees. This issue was addressed extensively in [7], where it was concluded that isotropic refinement is required if mesh quality is to be controlled effectively for arbitrary refinement levels. This is a serious limitation when directional flowfield features are present, leading to an inefficient distribution of grid points in the final mesh.

A remedy for this problem is to use hexahedral meshes, which do not have these element quality problems. This is because a hexahedron can be subdivided anisotropically in any of the three directions and yield child elements whose face angles are similar to their parent. This ability to anisotropically refine the mesh makes a tremendous difference in the final problem size when directional flow features are present.

Hexahedral meshes also have an advantage in that they yield more accurate flowfield solutions than their tetrahedral counterparts for the same number of edges. Aftosmis et al. [1] have shown that, in general, tetrahedral grids require approximately double the storage and CPU time than hexahedral tessellations of the same vertices. This is due to the fact that tetrahedral meshes have more edges. These additional edges, however, do not contribute significantly to an improvement in solution accuracy.

One disadvantage with unstructured hexahedral grids, however, is that grid generation is not as advanced as that for tetrahedral meshes. Some of this is due to the fact that this area has not received much attention till date. Fortunately, some hexahedral grid generation algorithms are now appearing in the literature [8, 12, 13, 14]. We realize that grid generation for hexahedral meshes is an important issue, but feel that the advantages of hexahedral mesh adaptation offset the current lack of highly-developed grid generation tools. We have therefore used structured hexahedral grids redefined in an unstructured manner as the initial meshes for the test problems reported in this paper.

[1] A leaf element is defined as one that has no children.

4 Hexahedral adaptation scheme

Our hexahedral adaptation scheme uses the same basic data structure and the edge-marking strategy as the tetrahedral scheme described in Section 2. Instead of a tetrahedral element being defined by its six edges, a hexahedral element is defined by its twelve edges. These edges are ordered in a particular manner such that the binary pattern of marked edges in each element determines how it will be refined. The twelve edges are grouped into three sets where each set consists of the four edges that are disjoint from one another. By handling each set of edges in succession, the refinement process is somewhat simplified and the total number of different ways an element can be subdivided is significantly reduced. Coarsening of hexahedral elements is also performed as in the tetrahedral scheme. If a child hexahedron has any of its edges marked for coarsening, this element and its siblings are removed and their parent element is reinstated with an adjusted edge-marking pattern. The parents are then subdivided based on their new patterns.

Hexahedral adaptation schemes generate "hanging" vertices when a hexahedron cannot be split into smaller hexahedra without continuously propagating the mesh refinement into regions where it is not desired. We solve this hanging-vertex problem by using other element types as buffers between refined and unrefined elements. In particular, pyramids and prisms are used to connect up the hanging vertices without unnecessarily propagating the grid refinement. These pyramids and prisms are never subdivided however. If an edge of a pyramid or a prism is marked for subdivision, then the element and its siblings revert back to their parent hexahedron and further refinement is performed directly on the hexahedral element itself. The basic logic for this treatment of buffer elements is identical to the scheme for tetrahedra that is described in [7] when the mesh quality option is turned on.

Sketches of the various types of allowed hexahedral element subdivisions are shown in Fig. 2. The refinement patterns in the top row occur when all four edges in any one or more of the three sets are bisected. This splits the original hexahedron into two, four, or eight smaller hexahedra. The child hexahedra thus have element quality properties that are similar to those of their parents. Note that all the four edges in a set are also bisected even if only three of them are marked for refinement. Similar action is also taken if only a pair of opposite edges in a set are marked.

At most one of the three sets can be marked non-uniformly; that is, only one set is allowed to have one edge or two adjacent edges marked for refinement. Note that in the current context, adjacent edges do not share a vertex but instead share a face. The non-uniformly marked set with only one bisected edge generates buffer pyramid elements and these are shown in the middle row of Fig. 2. Since there is only one set that has exactly one bisected edge, either the parent hexahedron or only one of its children is split into four pyramids.

The non-uniformly marked set with two adjacent bisected edges generates buffer prism elements and these are shown in the bottom row of Fig. 2. Depending on the actual marking pattern, either the parent hexahedron or at most two of its children are split into three prisms each. As mentioned earlier, the prisms and pyramids are never subdivided. They revert to their parent hexahedra if additional subdivision is required.

The Euler flow solver [4, 18] that is used with the tetrahedral elements is also used with these mixed-element meshes. Recall from Section 2 that the CFD solver is a vertex-

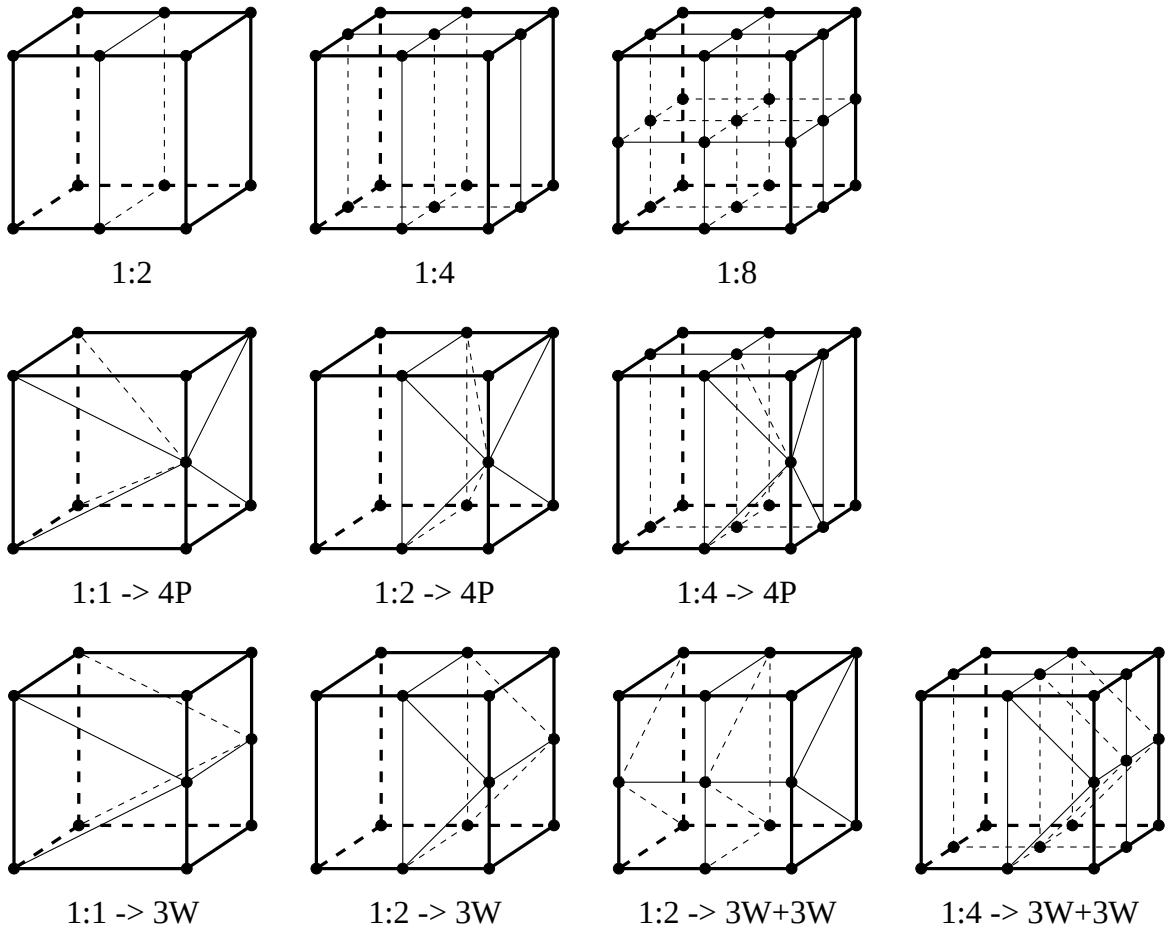


Figure 2: Types of subdivision that are permitted for a hexahedral element.

based scheme that satisfies the integral conservation laws on non-overlapping control volumes surrounding each vertex. Computation of the fluxes across the faces of the control volumes is carried out by summing the contributions from each edge in the mesh. These edges can make up arbitrary polyhedral elements. This means that flowfields on mixed-element meshes can be solved just as easily as those on purely tetrahedral grids.

One serious problem with this hexahedral adaptation scheme is the possibility of excessive propagation of the edge-marking process, and hence of the refinement region. As mentioned earlier, at most one of the three sets of edges can be marked non-uniformly. This restriction requires that elements be continuously upgraded if they have more than one set of edges that are marked non-uniformly. This causes mesh refinement to be propagated into regions where it is not desired.

Figure 3 depicts a scenario containing three contiguous elements where the refinement is propagated because the element edge-markings need to be upgraded to allowable patterns. The bottom element in the left stack has two edges marked for refinement, while the remaining two elements each have only one edge marked. Since only one set of edges can be marked non-uniformly, the bottom element must have all four edges in at least one of the two sets marked. This is shown in the middle stack. However, this causes the center element to inherit the problem. Using the same argument as before, the center element must now be

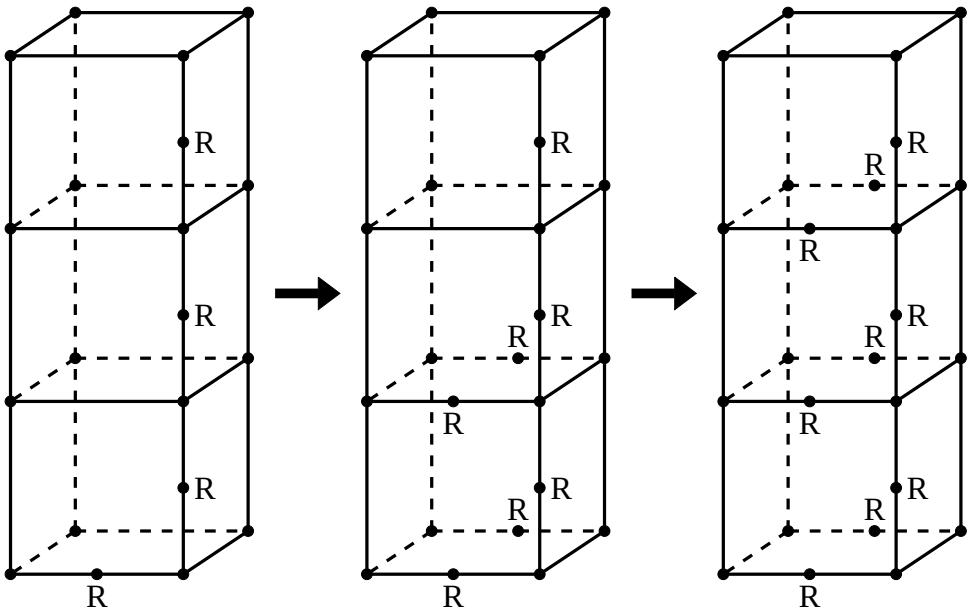


Figure 3: Schematic showing the problem of refinement propagation when more than one set of edges in a hexahedron are marked non-uniformly.

upgraded. This is shown in the third stack. At the next iteration, this process will continue as the top element will have to be upgraded. It is thus obvious that the refinement is being propagated unnecessarily due to the restriction that has been imposed.

A similar phenomenon of excessive propagation can also occur because of the requirement that all four edges in a set be bisected even if only three of the edges are marked for refinement. Figure 4 shows a scenario where this type of propagation occurs. Initially, only the bottom element has three edges in a set marked while the other two elements have two adjacent edges marked. But upgrading the bottom element to a 1:2 subdivision by remarking the fourth unmarked edge in the set causes the center element to have three marked edges in the same set. Upgrading the center element now will propagate the problem to the top element.

We solve this problem of refinement propagation by removing these requirements altogether. Only those hexahedral elements that will directly generate smaller hexahedra as dictated by their edge-marking patterns are first refined. Any partially-marked element is not upgraded but instead has a new vertex inserted at its center. This center vertex is then joined by edges to the eight vertices of the hexahedron, thereby creating six pyramids as shown in Fig. 5. It is possible that some of these pyramids have large angles at the center vertex. This is unavoidable without performing some sort of mesh smoothing. However, this deterioration in mesh quality is bounded since these pyramids are never refined more than once.

Each of these six pyramids are then examined sequentially. Each pyramid has a face of the original hexahedron as its base. The four edges that constitute this face can have one of six distinct refinement patterns. This causes the special subdivisions that are shown in Fig. 6. If none of the edges on this face are marked for refinement, the pyramid element remains unchanged. If only one edge is marked, three tetrahedral elements are generated.

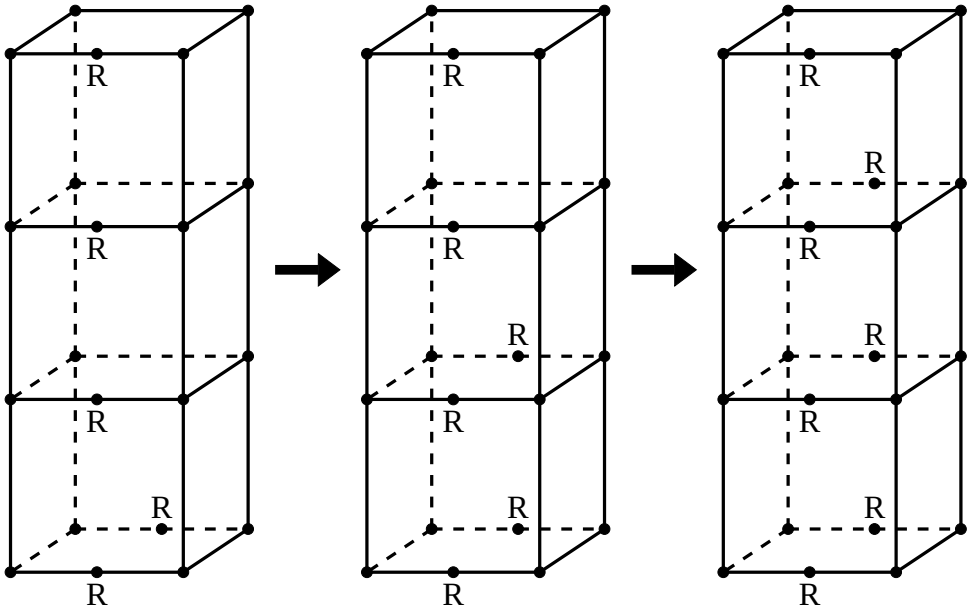


Figure 4: Schematic showing the problem of refinement propagation when only three edges of a set are marked.

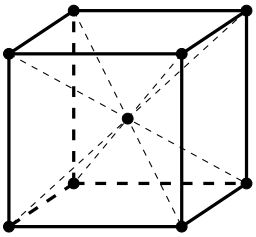


Figure 5: Six pyramid elements are initially created by inserting a vertex at the center of a marked hexahedron.

If two adjacent edges are marked, four tetrahedral elements are obtained. However, if two opposite edges are marked, then two pyramid elements are created. If three edges are marked for refinement, then we generate one pyramid and three tetrahedral elements. Finally, if all the four face edges are bisected, we obtain four smaller pyramids.

Insertion of the center vertex obviously eliminates all possibility of refinement propagation. No additional edges are ever marked for subdivision and none of the neighboring hexahedral elements are affected. The subdivision process is thus always performed locally within the marked hexahedra which are then properly split into a combination of smaller hexahedral, pyramid, and tetrahedral elements to obtain a valid mesh connectivity.

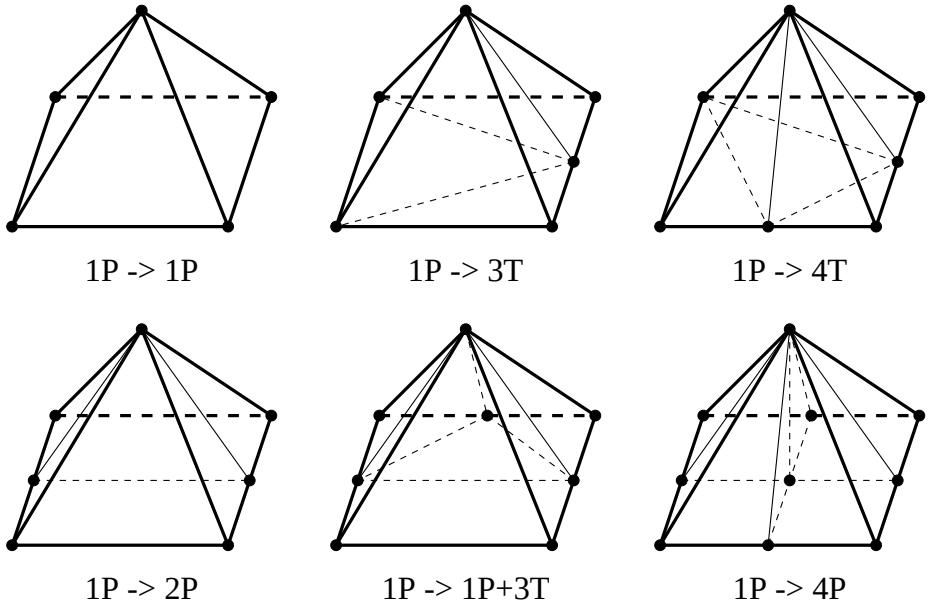


Figure 6: Special subdivision types when inserting a vertex at the center of a hexahedron.

5 Computer resources

The computer resources required for the tetrahedral and hexahedral mesh adaptation algorithms depend on several factors [6]. Because of this complexity, there is no general expression for the CPU time in terms of the problem size. However, computer memory requirements can be readily expressed as a certain number of integers per vertex, and CPU times are quoted for a general, realistically-sized test case.

For direct comparison between the two adaptation schemes, assume that both meshes have N vertices. The asymptotic estimates for the number of elements and edges for tetrahedral meshes [3] are $5N$ and $6N$, respectively, but are N and $3N$ for hexahedral meshes. When combined with the well-known asymptotic estimate that the number of faces is twice the number of elements, we have $10N$ and $2N$ faces for tetrahedral and hexahedral meshes, respectively.

With the above assumptions, our tetrahedral adaptation scheme requires 160 integers per vertex beyond the storage that is directly required by the flow solver. In contrast, the hexahedral scheme requires only 70 integers per vertex. This is because our data structures are based on edges and a hexahedral tessellation has only half as many edges as a tetrahedral tessellation of the same distribution of vertices. The storage requirement numbers do not include the overhead for retaining the parent elements and edges. This overhead increases with the number of refinement levels and the fraction of the mesh that is refined; however, it can never be more than 15% of the baseline requirements.

As mentioned earlier, there is no general expression for the CPU time requirements for mesh adaptation; we can only report actual times from test runs. For example, when about 20% of the mesh is adapted, the tetrahedral scheme requires 200 μsecs per vertex whereas the hexahedral scheme requires 130 μsecs on a single-processor Cray C-90. This reduction in time for the hexahedral case is expected because most of the adaptation time for both

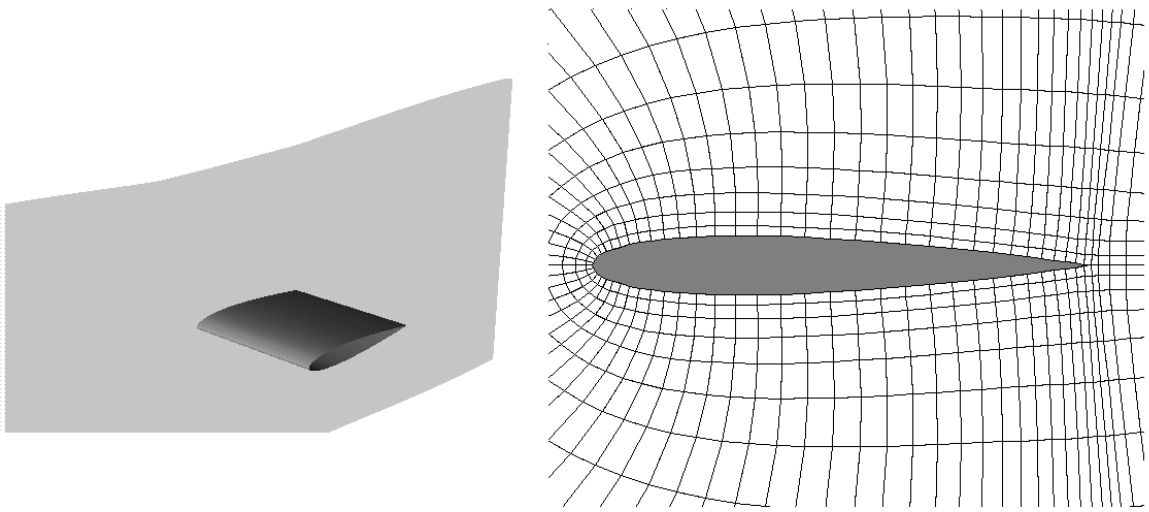
schemes is spent looping over the list of edges.

Recall that the Euler flow solver [18] that we use for our calculations is a cell-vertex scheme in three dimensions. As a result, the number of flux calculations is proportional to the number of edges in the mesh. Explicit time evolution is done on control volumes surrounding each vertex; hence, the computational effort is proportional to the number of vertices. This solver requires approximately 250 integers per vertex for tetrahedral meshes and 180 integers per vertex for hexahedral meshes. It's C-90 CPU time requirements are 65 μsecs and 35 μsecs per vertex per time step for the tetrahedral and hexahedral meshes, respectively. Thus, our mesh adaptation schemes need between three and four times the amount of CPU time required for one time step of the Euler flow solver on a C-90 when about 20% of the mesh is adapted. This is acceptable considering that the adaptation codes do not vectorize and work primarily with integer arrays; thus, they do not utilize two of the salient features of the C-90 (which is a 64-bit vector supercomputer).

6 Computational results

Both the tetrahedral and the hexahedral mesh adaptation procedures have been applied to two transonic flow problems. Even though both these test cases represent steady-state problems, the algorithms are applicable to unsteady flows as well. Steady-flow examples have been chosen as the simplest test cases that fully exercise all aspects of the adaptation schemes in three dimensions.

The first test problem is a NACA 0012 wing that is mounted between two inviscid sidewalls. This is shown in Fig. 7(a). The advantage of this case is that although the problem is three-dimensional, the results can be easily visualized along the sidewalls and compared to



(a) (b)

Figure 7: Views of (a) a NACA 0012 wing mounted between two inviscid sidewalls and (b) a close-up of the initial hexahedral mesh.

existing high-resolution two-dimensional computations. Flow conditions for this case were a free-stream Mach number of 0.85 and an one-degree angle of attack.

The initial three-dimensional computational mesh was created from two structured C-H meshes placed one chordlength apart in the spanwise direction. The mesh consisted of 4006 vertices and 1900 hexahedral elements. A total of 69 points were located on the surface of the wing at each two-dimensional plane. The outer computational boundaries were located at approximately 20 chords above and below the wing. A view of the initial hexahedral mesh is shown in Fig. 7(b). A tetrahedral mesh was generated by splitting each hexahedron into five tetrahedra.

Results for the tetrahedral adaptation scheme are not presented here as they have been reported and analyzed in earlier work [6]. Figure 8 shows two hexahedral mesh adaptation steps for this problem. The absolute difference in density across each edge of the mesh was used as the error indicator. Approximately 4000 edges were targeted for refinement the first time. A total of 5038 edges were then coarsened and 4972 edges refined. These values were chosen only to obtain a reasonable solution for the problem. No attempt was made to optimize the adapted mesh for efficiency or the accuracy of the final computed results. The final mesh consisted of 14,120 vertices, 36,886 edges, 5942 hexahedral elements, and 1911 buffer elements. The flow solver was run for approximately 500 iterations on each intermediate mesh. The fact that an adapted mesh starts out with the interpolated coarse-grid solution means that it converges rapidly for steady-state calculations, even on fine meshes.

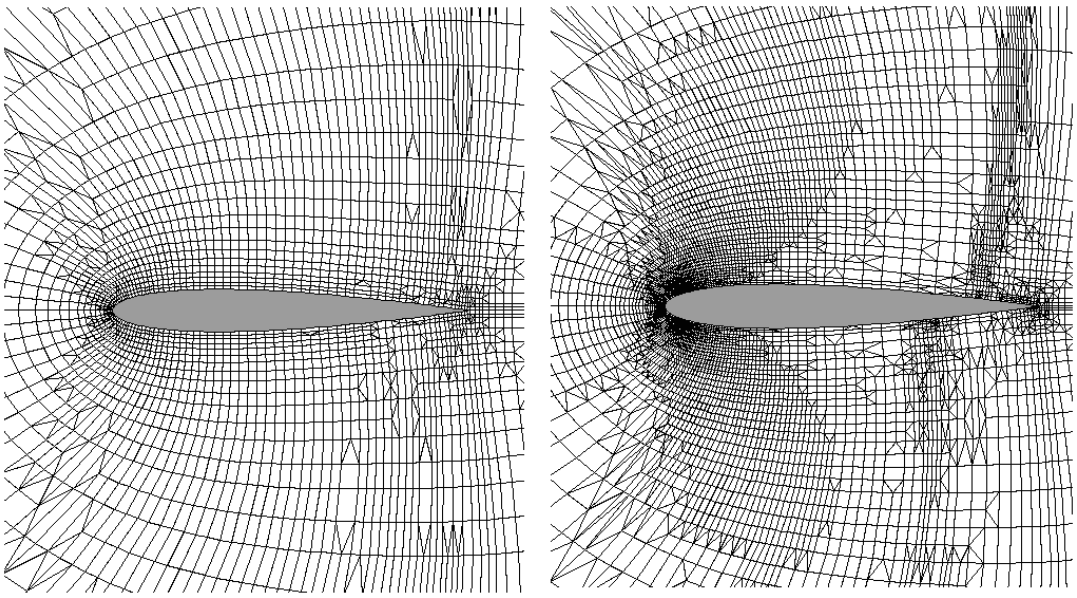


Figure 8: Two mesh adaptation stages for the inviscid NACA 0012 wing.

Pressure contours for this case are shown in Fig. 9. These contours show good resolution of the shock on the upper surface out to the far field. Additional mesh adaptation steps will help to sharpen up the shock on the lower surface. The stagnation point at the leading edge is also captured with high resolution. Results for the computed pressure coefficient

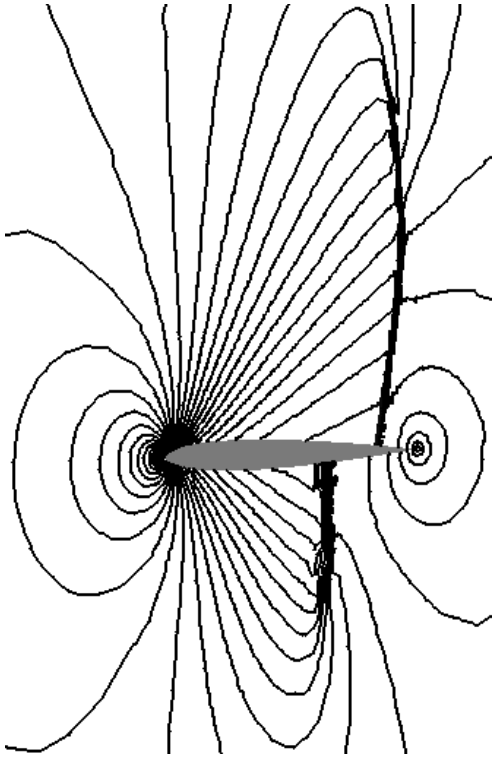


Figure 9: Computed pressure contours at the sidewalls for the solution-adapted hexahedral inviscid wing mesh.

on the final mesh are presented in Fig. 10. These results are compared to the AGARD Euler solution No. 9 taken from [2]. Their two-dimensional structured-grid solution used an O-mesh consisting of 320 points on the airfoil surface and 64 points normal to the surface. The outer computational boundary was located 25 chords from the airfoil surface. The results in Fig. 10 show excellent agreement between the two solutions. We expect that the slight difference in the shock location on the lower surface will disappear with a better error indicator and more levels of adaptation. Similar results were obtained with the tetrahedral adaptation scheme [6]. The purpose of this test case is to show that the solution-adaptive unstructured grid calculations have a more efficient mesh distribution than the structured-grid counterpart.

The second test case is a non-lifting rectangular helicopter rotor blade in hover. It simulates the model-rotor acoustics experiment of Purcell [15] where a 1/7th scale model of a UH-1H rotor blade was tested over a range of hover-tip Mach numbers from 0.85 to 0.95. These rotor speeds produce strong aerodynamic shocks at the blade tips and an annoying pattern of high-speed impulsive noise. Since the model-rotor blades were untwisted and run in a non-lifting configuration, the problem was symmetric about the rotor plane and eliminated the need to model the wake system. Results in this paper are only presented for the 0.95 hover-tip Mach number case.

The computational mesh only needs to cover the upper half plane because of the symmetry condition of the non-lifting configuration. A conventional C-H structured grid extending out to two rotor radii in the spanwise direction was first generated. Grid points off the blade

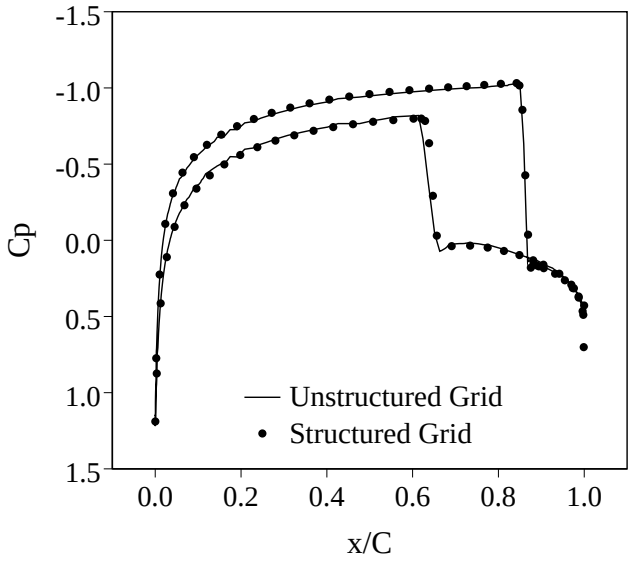


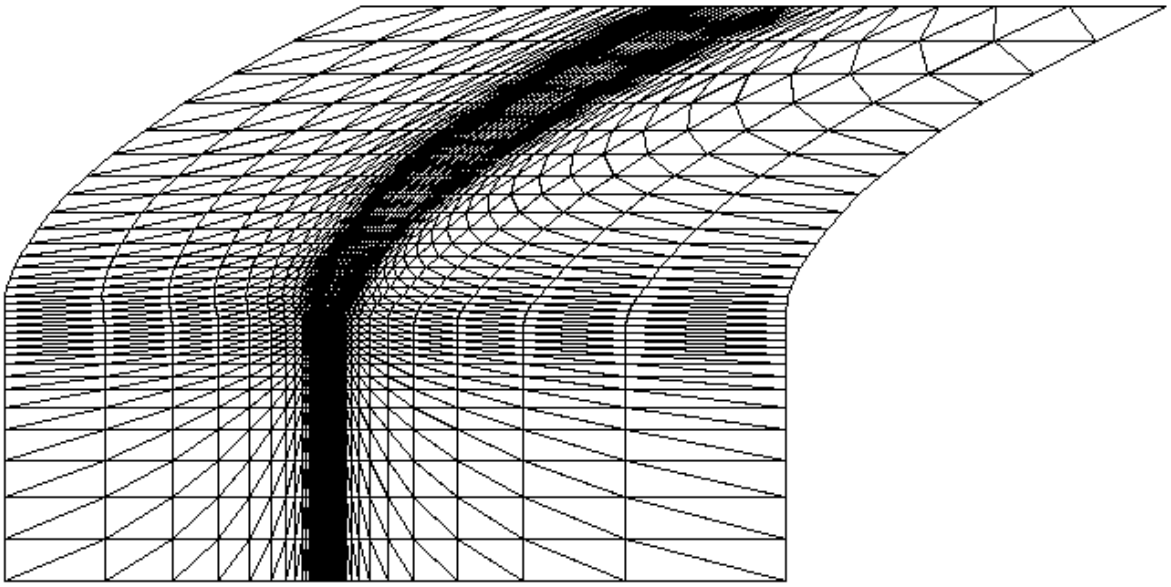
Figure 10: Computed surface pressures from the solution-adaptive hexahedral mesh are compared to those from a structured grid.

tip were concentrated along the expected path of the acoustic wave. An unstructured hexahedral grid was then created by redefining the structured grid in an unstructured manner. A tetrahedral grid was generated by splitting each hexahedron into five tetrahedra. These meshes are shown in Fig. 11 and served as base meshes for the initial solutions and subsequent local mesh adaptations. The rotor blade has an aspect ratio of 13.71 and a NACA 0012 cross-section. Both initial meshes had 25,520 vertices and extended 1.5 rotor radii above the rotor plane. The tetrahedral mesh had 114,589 elements and 145,142 edges. The hexahedral mesh was much smaller: it had only 22,895 elements and 73,766 edges.

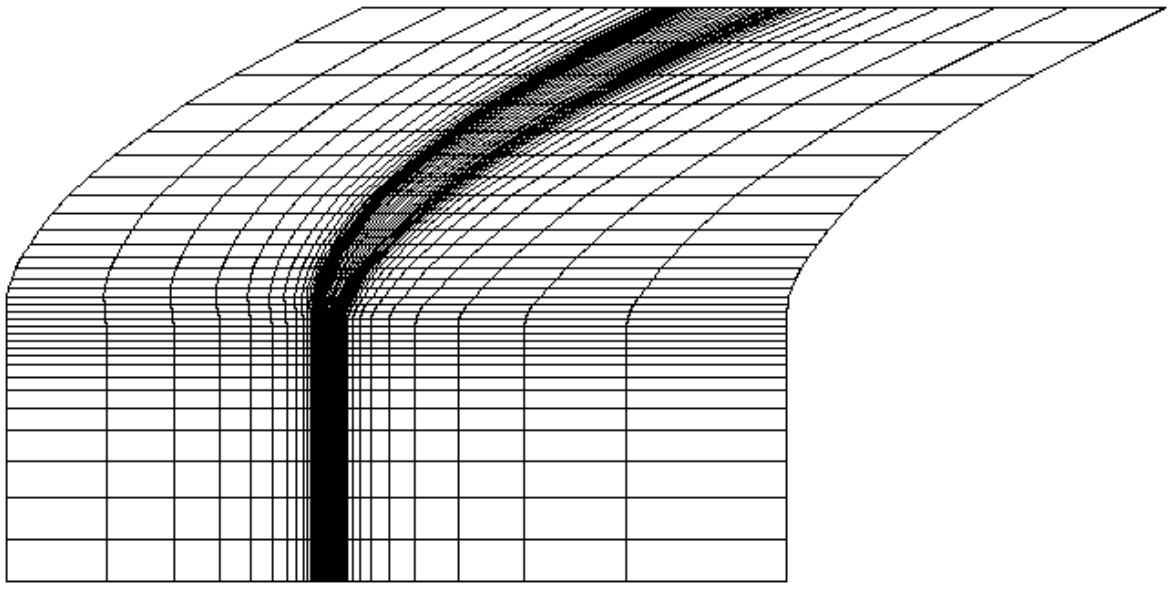
Initial solutions were first obtained on these grids by running the flow solver for 1000 time steps. Computed surface pressures at the 95%-span station for the two meshes are compared in Fig. 12(a). Results indicate that the hexahedral mesh has a sharper shock on the blade surface. A similar improvement is observed in Fig. 12(b) where the acoustic pressures at 1.2 rotor radii are compared. These results demonstrate that the additional edges in the tetrahedral mesh did not contribute to an improvement in the solution accuracy. In fact, the hexahedral mesh yielded better solutions using fewer edges and the same vertex distribution.

Both the meshes shown in Fig. 11 were then adapted using an error indicator based on acoustic pressure. A detailed description of the error indicator is given in [19]. A total of three refinement and two coarsening steps were performed on the tetrahedral mesh. The flow solver was run for about 500 time steps between mesh adaptations. A close-up of the final tetrahedral mesh and pressure contours in the rotor plane are shown in Fig. 13. The mesh has been refined to adequately resolve the leading edge compression and capture both the surface shock and the resulting acoustic wave that propagates to the far field. Extensive details of this acoustic calculation are given in [19]. The final mesh contained 140,419 vertices, 933,855 edges, and 783,164 tetrahedra.

The initial hexahedral mesh was refined similarly. The final mesh after two adaptations is shown in Fig. 14. The mesh was almost quadrupled in size to 103,772 vertices, 343,411 edges,



(a)



(b)

Figure 11: Initial (a) tetrahedral and (a) hexahedral meshes in the plane of the helicopter rotor.

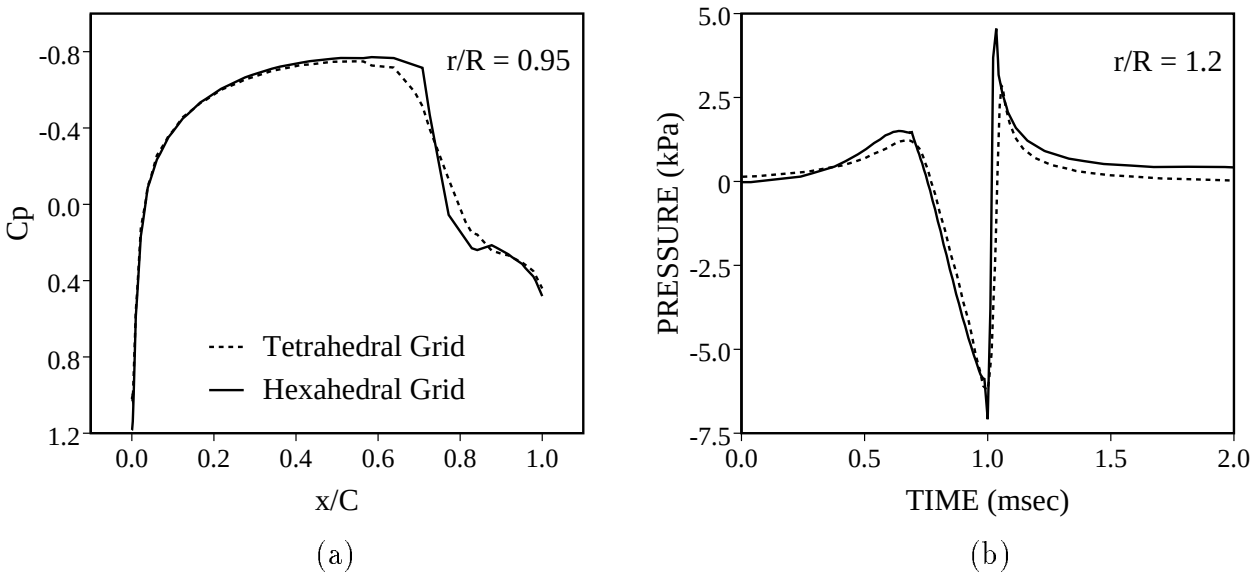


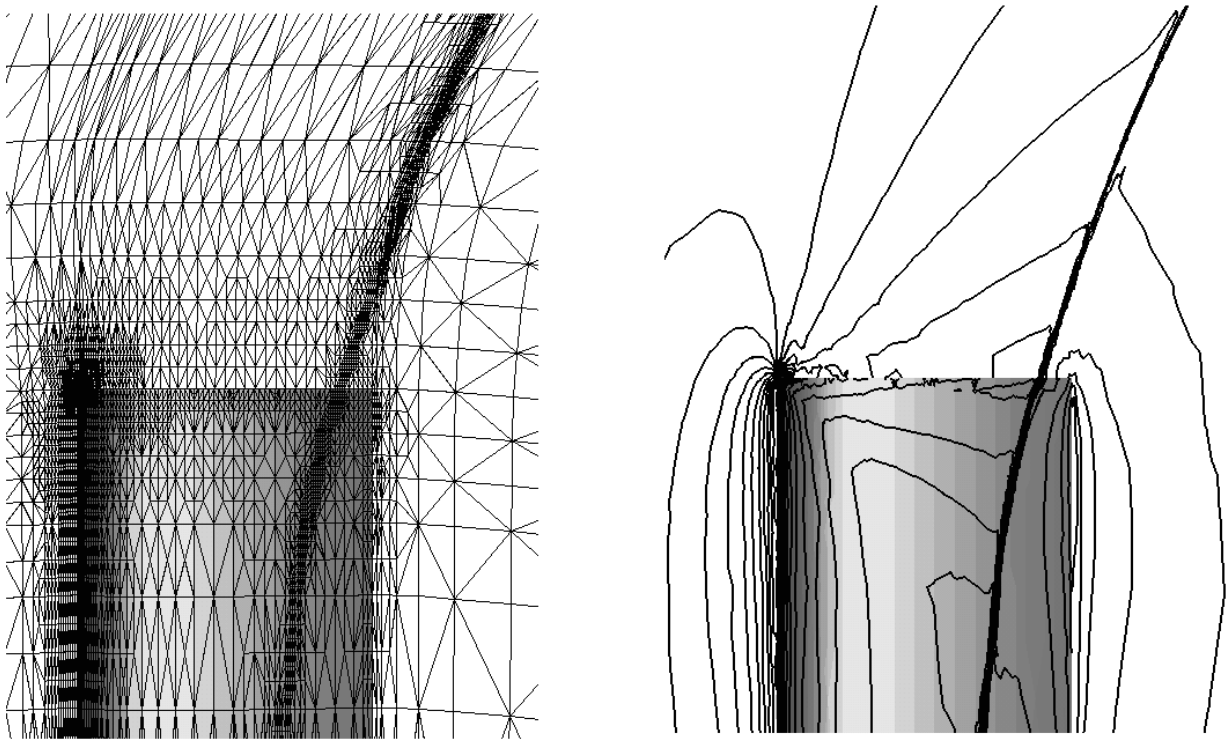
Figure 12: Comparison of solution quality on the initial tetrahedral and hexahedral meshes: (a) surface pressure at 95% spanwise location and (b) acoustic pressure at 1.2 rotor radii in the rotor plane.

75,977 hexahedral elements, and 59,143 buffer elements. Additional levels of adaptation are needed to adequately capture the acoustic wave. No direct comparisons similar to those shown in Fig. 12 were done between the adapted tetrahedral and hexahedral meshes as it is difficult to be unbiased.

7 Summary

This paper has presented two methods for the dynamic adaptation of three-dimensional unstructured meshes. Both the tetrahedral and the hexahedral procedures use edge-based data structures to allow anisotropic refinement and coarsening. Hexahedral meshes have the advantage that they can be repeatedly subdivided anisotropically without deteriorating the element quality. However, to eliminate hanging vertices in the mesh, pyramids, prisms, and tetrahedra are used as buffer elements to interface regions of refined and unrefined hexahedra. A problem with excessive propagation of the refinement region for hexahedral meshes is eliminated by not upgrading element edge-marking patterns but by inserting a vertex at the center of the hexahedron instead. The hexahedron is then locally split to generate a valid mesh. A comparison of the computer resources that are required for both types of grids have shown that hexahedral meshes have half the storage requirements and run almost twice as fast as tetrahedral meshes having the same distribution of vertices.

The solution-adaptive schemes have been demonstrated for two sample CFD cases. Computed solutions for the Euler equations on a psuedo-three-dimensional problem showed good agreement with results from a conventional structured-grid method. Results were also presented for a realistic test case in helicopter acoustics that showed these methods can be applied efficiently to large CFD problems. Results also indicated that hexhedral meshes



(a) (b)

Figure 13: Final (a) tetrahedral mesh and (b) pressure contours in the rotor plane.

utilize computer resources more efficiently than tetrahedral meshes for the same level of solution accuracy.

References

[1] M. Aftosmis, D. Gaitonde, and T.S. Tavares, On the accuracy, stability, and monotonicity of various reconstruction algorithms for unstructured meshes, AIAA-94-0415, 32nd AIAA Aerospace Sciences Meeting (1994).

[2] AGARD Fluid Dynamics Panel Working Group 07, Test cases for inviscid flow field methods, AGARD-AR-211 (1985).

[3] T.J. Barth, Numerical aspects of computing viscous high Reynolds number flows on unstructured meshes, AIAA-91-0721, 29th AIAA Aerospace Sciences Meeting (1991).

[4] T.J. Barth, A 3-D upwind Euler solver for unstructured meshes, AIAA-91-1548, 10th AIAA Computational Fluid Dynamics Conference (1991).

[5] C.W. Berry, S.R. Kennon, and T.J. Liszka, Anisotropic h-refinement and unrefinement for hexahedral finite elements, 3rd U.S. National Congress on Computational Mechanics (1995) 163.

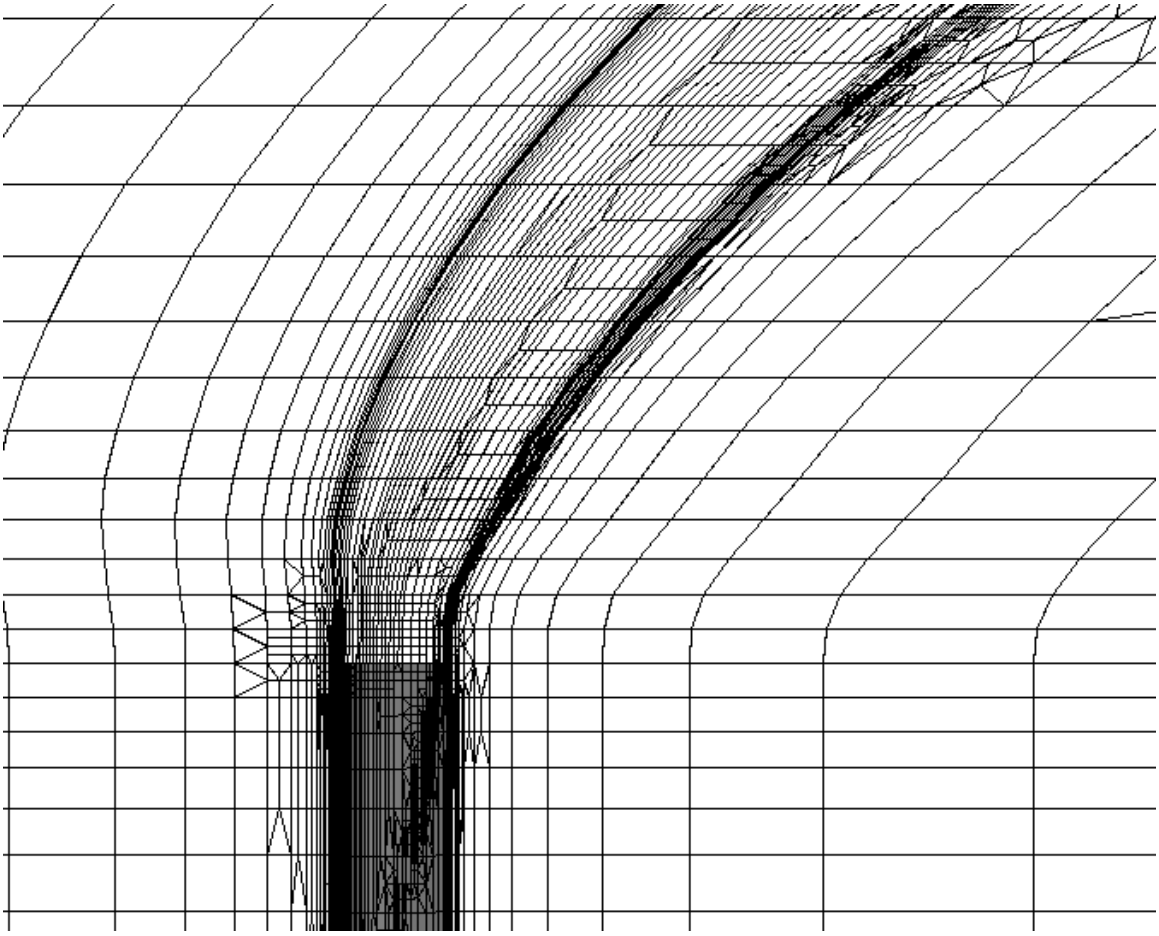


Figure 14: Final hexahedral mesh in the rotor plane.

[6] R. Biswas and R.C. Strawn, A new procedure for dynamic adaption of three-dimensional unstructured grids, *Appl. Numer. Math.* 13 (1994) 437–452.

[7] R. Biswas and R.C. Strawn, Mesh quality control for multiply-refined tetrahedral grids, *Appl. Numer. Math.* 20 (1996) 337–348.

[8] T.D. Blacker and R.J. Meyers, Seams and wedges in plastering: A 3-D hexahedral mesh generation algorithm, *Eng. with Comp.* 9 (1993) 83–93.

[9] C.L. Bottasso, H.L. deCougny, M. Dindar, J.E. Flaherty, C. Ozturan, Z. Rusak, and M.S. Shephard, Compressible aerodynamics using a parallel adaptive time-discontinuous Galerkin least-squares finite element method, AIAA-94-1888, 12th AIAA Applied Aerodynamics Conference (1994).

[10] Y. Kallinderis and P. Vijayan, An adaptive refinement/coarsening scheme for 3-D unstructured meshes, *AIAA J.* 31 (1993) 1440–1447.

[11] R. Löhner and J.D. Baum, Adaptive h-refinement on 3D unstructured grids for transient problems, *Int. J. Numer. Methods Fluids* 14 (1992) 1407–1419.

[12] D.J. Mavriplis, Adaptive meshing techniques for viscous flow calculations on mixed element unstructured meshes, AIAA-97-0857, 35th AIAA Aerospace Sciences Meeting (1997).

[13] M.A. Price and C.G. Armstrong, Hexahedral mesh generation by medial surface subdivision: II. Solids with flat and concave edges, *Int. J. Numer. Methods Eng.* 40 (1997) 111–136.

[14] M.A. Price, C.G. Armstrong, and M.A. Sabin, Hexahedral mesh generation by medial surface subdivision: I. Solids with convex edges, *Int. J. Numer. Methods Eng.* 38 (1995) 3335–3359.

[15] T.W. Purcell, CFD and transonic helicopter sound, Paper No. 2, 14th European Rotorcraft Forum (1988).

[16] R. Rausch, J. Batina, and Y. Yang, Spatial adaptation procedures on tetrahedral meshes for unsteady aerodynamic flow calculations, AIAA-93-0670, 31st AIAA Aerospace Sciences Meeting (1993).

[17] R. Schneiders and J. Debye, Refining quadrilateral and brick element meshes, in: I. Babuska et al., eds., *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, IMA Vol. 75 (Springer-Verlag, New York, 1995) 53–65.

[18] R.C. Strawn and T.J. Barth, A finite-volume Euler solver for computing rotary-wing aerodynamics on unstructured meshes, *J. AHS* 38 (1993) 61–67.

[19] R.C. Strawn, R. Biswas, and M. Garceau, Unstructured adaptive mesh computations of rotorcraft high-speed impulsive noise, *J. Aircraft* 32 (1995) 754–760.

[20] J.J.W. Van Der Vegt, Anisotropic grid refinement using an unstructured discontinuous Galerkin method for the three-dimensional Euler equations of gas dynamics, AIAA-95-1657, 12th AIAA Computational Fluid Dynamics Conference (1995).